**EMON Data Processing (EDP) Tool**

**Version 4.23**

February 16, 2021

**Features**

EMON Data Processing (EDP) tool is designed to facilitate both collection and processing of EMON data across both steady and unsteady state workloads. This tool has the following features

* Provides a list of critical emon events that are needed for in-depth characterization across many architectures (Haswell, Broadwell, Sklylake, Cascadelake,..)
  + Users can add/remove events to suit their needs
  + Users can easily add support for any new architecture
* Pre-defined list of several important metrics (like CPI, MPI, memory b/w etc)
  + Users can add their own list of metrics as well
* Provides post-processing scripts to consume the emon data collected
  + Can limit the analysis to a specific range of samples collected to support workloads with multiple phases
* Automatically computes average, min, max, variances of samples and also plots time-series graphs in excel
  + Full control over plotting any specific metric or emon event

**Usage**

This tool depends on SEP/EMON driver to collect the raw performance counters from the hardware and requires SEP/EMON driver installation on the system prior to using EDP. EDP package provides the specific set of events that are needed to be collected from the hardware. These events are used by the SEP/EMON driver to collect the data. Once the data is collected, post processing is done. For this phase, EDP provides an XML file that has all the metrics and corresponding formula to compute those using various emon events.

**Contents**

The ZIP file contains the following files

1. Edp.rb – Main script that post-processes the EMON data
2. process.cmd - Batch file to tailor the post-processing. This invokes edp.rb script. (please remove .rename extension before use, if present)
3. jprocess.cmd – The previous batch file does sequential processing. If the input is large, parallel processing may be preferred (assuming you have a lot of h/w threads on your system where post-processing is taking place). You can choose to use either of these .cmd files as they contain largely similar configuration information but for extra jRuby configurations.
4. HybridProcess.cmd – Batch file to tailor post processing, analogous to process.cmd, for hybrid systems. This invokes the edp.rb script once for each type of processors on the platform, yielding separate output files for each.
5. Architecture-specific – This directory contains folders for all the supported architectures. Within each architecture folder, multiple folders exist for 1S/2S/4S/8S configurations. In each of those folders, you can see the following 3 files (names dependent on architecture selected)
   1. clx-2s.xml : contains metrics and the formula that are required. This file needs to be copied to the directory where process.cmd is installed.
   2. clx-2s-events.txt : emon event file that is used to collect emon data. This file needs to be copied to the server where emon data needs to be collected.
   3. Chart\_format\_clx-2s.txt : Defines all the metrics that needs to have time-series charts in excel. This file needs to be copied to the directory where process.cmd is installed.

**EMON Data Collection Steps**

1. **The processor specific EMON script (like clx-2s-events.txt, skx-2s-events.txt etc) that is provided collects EMON events at a sampling frequency of 0.1 sec. The loop count (-l parameter) is set to a large value. Launch emon data collection with the command “*emon –i clx-2s-events.txt > emon.dat***
   1. **If desired, you can change the output file name to a more meaningful name**
   2. **Launch the script as part of launching the workload as we want to collect the events over the entire duration of the workload execution. (unless you know exactly which portion of the workload you are interested in characterizing)**
2. **Since EMON collection is set to a large loop count, it will run for a long time and needs to be stopped as soon as the workload has completed. This can be accomplished by using emon –stop command to stop the collection any time on all o/s**

**EMON Data Post Processing Steps**

EDP tool post-processes the EMON data and produces various metrics like LLC MPI, CPI, memory b/w etc

1. This tool is based on Ruby and runs on Windows platform. The tool can automatically launch excel and import the data and draw charts as well. You can download the latest ruby from <https://www.ruby-lang.org/en/downloads/>. Please look at the last section titled “How to Post Process on Linux” for processing on Linux.
2. Extract the ZIP file emon-post-processing-tool-\*.zip file in a directory. Remove .rename extensions from process.cmd and the shell script.
3. The post-processing tool package also includes 3 architecture specific input files that need to be copied from the “architecture Specific” sub folder in the ZIP file to the directory where process.cmd file is stored.
4. Process.cmd file needs to be modified by the user as it has some hard-coded names (like emon.dat etc). If those names are different in your environment, please do change them.
5. Since EMON data is collected over the entire duration, you may not want to look at metrics over the entire duration (as it may include start-up and other ramp-up activities). The tool provides a way to specify begin and end sample numbers over which the metrics will be aggregated.
6. By default only the system view is generated. The other three views (socket, core and thread) can be selected optionally.
7. Copy emon.dat file collected from the target system and launch process.cmd. The output will be a excel spreadsheet named summary.xlsx
   1. In case you don’t have Excel installed on your system, you can use the .csv file created to look at the data.
8. The summary.xlsx file contains 4 groups of tabs named ‘\* view’, ‘\* view (per-txn)’, ‘chart \* view’, ‘details \* view’.
   1. ‘system view’, ‘socket view’, ‘core view’ and ‘thread view’ are the summary data for each view respectively.
   2. system view (per-txn)’, ‘socket view (per-txn)’, ‘core view (per-txn)’ and ‘thread view (per-txn)’ are the per-transaction version of the summary data. They are available only when the throughput (TPS) is specified in command line by --tps parameter in process.cmd or jprocess.cmd file. In these views, some of the performance metrics are normalized to per-transaction, instead of per-instruction as displayed in the ‘\* view’ views. For those metrics that cannot be normalized to per transaction, they are left unchanged. All the raw events are normalized to per transaction as well.
   3. Summary tab contains details like average, min/max etc for various metrics and events over the collection period (or between begin and end sample period if specified in process.cmd). In this tab, the 1st column contains the names. The next column named “aggregated” is the sum of all events across all samples divided by either sum of all clocks or sum of other event across all those same samples. In previous versions of EDP (1.4k or older) another column named “Average” was provided that looks at averages computed based on each individual sample (it is more like average of averages). For example, CPI will be calculated on each individual sample group and the average of all those CPI values will be presented. Whereas in case of aggregated, we will add all cycles across all samples and divide that by the sum of all instructions retired in all those samples. In case you still want to see the “average” column in 1.4m or newer versions of EDP, please add the flag “--showaverage” to process.cmd file.
   4. Min, max and stddev are also provided for the samples selected in the process.cmd script. The last column provides a measure of the variation in the data across different samples. Available only in system view
   5. Max\_95% column shows 95th percentile value for any particular event or metric. This is very useful to discard occasional huge spikes. Available only in system view
   6. All the values shown are ***normalized per second*** unless otherwise stated (as in misses per instruction)
   7. The ‘chart \* view’ tabs have time-series charts for all the metrics and events.
   8. The ‘details \* view’ tabs have details data used for generating summary and charts. Note those data have been normalized per second too.

**Parallel Data Processing**

Since version 1.2, EDP supports parallel data processing, i.e., using multiple threads to speed up the data processing. EDP is developed using Ruby programing language. Although Ruby does support multi-threading programming, the reference implementation of Ruby (what you can find at <http://www.ruby-lang.org/en/>) does not allow more than one of the threads to run at the same time, because some of the C libraries used in this implementation are not themselves thread-safe. Fortunately there is an alternative Ruby implementation, JRuby, which is based on Java and support real multi-threading. However, there is a known bug in JRuby’s win32ole library, which will cause Excel spreadsheet generation to fail. So as a workaround, we just split the tool into two steps: the first step uses JRuby for better performance, and the 2nd step uses CRuby for stability. In future, those issues will be fixed and either one of the JRuby and CRuby should work.

Steps:

1. Download JRuby from: <http://jruby.org.s3.amazonaws.com/downloads/1.6.1/jruby_windows_x64_jre_1_6_1.exe>
2. Edit the jprocess.cmd, and adjust the following environment variables:
   1. JRUBY: where the jruby binary locates
   2. JRUBY\_OPTIONS: the JVM options. Usually you just need to change the heap size to an appropriate value.
   3. PARALLELISM: the number of threads to process data in parallel.
3. Run jprocess.cmd to kick off the processing.

**Hybrid Data Processing**

Since version 4.23, EDP supports Hybrid EMON data analysis, using HybridProcess.cmd. The setup and usage follows that of post processing with process.cmd, with the following exceptions.

1. The post-processing tool package includes 3 architecture specific input files, per each core type, that need to be copied from the “architecture Specific” sub folder in the ZIP file to the directory where HybridProcess.cmd file is stored.
2. HyrbidProcess.cmd file needs to be modified by the user as it has some hard-coded names (like emon.dat etc). If those names are different in your environment, please do change them.
3. Copy emon.dat file collected from the target system and launch HybridProcess.cmd. The output will be several excel spreadsheets prefixed with summary\_, ending with the core type name.

**Customizing EDP tool**

EDP tool is highly customizable. You can define your own set of events to be collected, define your own metrics and also limit the metrics that are charted using excel.

**Customizing Events collected**

* For each architecture, we have provided an event file with a list of events that are critical for characterization. The number of events in the list is intentionally kept small as more event groups lead to fewer samples for each event leading to loss of accuracy. You can add/remove events from this list and use that as an input for data collection. By default, collection scripts are meant for internal use and include private emon events. For some processors, a sub-directory called “public version” is provided that can be used to collect public events and can be shared with external customers.

**Customizing metrics**

* A convention to name all the metrics with a prefix of metric\_\* has been followed here. However, the tool does not mandate that. We have provided metrics (along with formula) based on the events defined in an XML file (like wsm-ep.xml etc). This file contains entries like this

<metric name=**"metric\_memory\_bandwidth\_total (MB/sec)"**>

<event alias=**"a"**>**UNC\_IMC\_NORMAL\_READS.ANY**</event>

<event alias=**"b"**>**UNC\_IMC\_WRITES.FULL.ANY**</event>

<formula>**(a+b)\*64/1000/1000**</formula>  
</metric>

* System’s parameters can be referred in metric formula as *constant*. For example, **system.sockets[0].cores[0].count** means the number of CPUs in socket\_0/core\_0.

<metric name=**"metric\_L2\_Core sends HITM response to snoop (per inst)"**>

<event alias=**"a"**>**SNOOP\_RESPONSE.HITM**</event>

<constant alias=**"b"**>**system.sockets[0].cores[0].count**</constant>

<event alias=**"c"**>**INST\_RETIRED.ANY**</event>

<formula>**a/b/c**</formula>

</metric>

The table below lists some constants that can be used in formula definition. Note they are case sensitive.

|  |  |
| --- | --- |
| Name | Description |
| system.tsc\_freq | TSC frequency in Hz |
| system.qpi\_freq | QPI frequency in T/s |
| system.cpus.count | # of CPUs in the system |
| system.sockets.count | # of sockets in the system |
| system.sockets[0].cores.count | # of cores in socket 0 |
| system.sockets[0].cores[0].count | # of threads in a socket 0 / core 0 |
| system.sockets[0].cpus.count | # of logical cpus in socket 0 |

* A pseudo-event TSC is defined, which can be used in some metrics as if there is a real TSC event for each CPU. Note it is different between **CPU\_CLK\_UNHALTED.REF/TSC** and **CPU\_CLK\_UNHALTED.REF/system.tsc\_freq**: when calculating per socket/system value, the former leads to sum\_of\_all\_cpus(CPU\_CLK\_UNHALTED.REF) / sum\_of\_all\_cpus(TSC), while the latter leads to sum\_of\_all\_cpus(CPU\_CLK\_UNHALTED.REF) / constant\_value\_tsc\_freq.

<metric name=**"metric\_CPU\_utilization"**>

<event alias=**"a"**>**CPU\_CLK\_UNHALTED.REF**</event>

<event alias=**"b"**>**TSC**</event>

<formula>**a/b**</formula>

</metric>

* By default metrics are calculated from the events on local socket. However, some per-socket metrics need to be calculated from events on remote socket. In such case, per-socket formula can be defined and events on individual socket can be referred with a subscript. In the example below, i[0] means the aggregated value of the corresponding event on socket 0. For uncore events that have multiple values for each socket, the values will be aggregated too.

<metric name=**"metric\_NUMA\_%\_Reads satisfied by local DRAM"**>

<event alias=**"a"**>**UNC\_SNP\_RESP\_TO\_LOCAL\_HOME.FWD\_S\_STATE**</event>

<event alias=**"b"**>**UNC\_SNP\_RESP\_TO\_LOCAL\_HOME.FWD\_I\_STATE**</event>

<event alias=**"c"**>**UNC\_SNP\_RESP\_TO\_LOCAL\_HOME.WB**</event>

<event alias=**"d"**>**UNC\_SNP\_RESP\_TO\_REMOTE\_HOME.FWD\_S\_STATE**</event>

<event alias=**"e"**>**UNC\_SNP\_RESP\_TO\_REMOTE\_HOME.FWD\_I\_STATE**</event>

<event alias=**"f"**>**UNC\_SNP\_RESP\_TO\_REMOTE\_HOME.WB**</event>

<event alias=**"g"**>**UNC\_QHL\_REQUESTS.REMOTE\_READS**</event>

<event alias=**"i"**>**UNC\_QHL\_REQUESTS.LOCAL\_READS**</event>

<formula>**100\*(i-d-e-f)/(i+g)**</formula>

<formula socket=**"0"**>**100\*(i[0]-d[1]-e[1]-f[1])/(i[0]+g[1])**</formula>

<formula socket=**"1"**>**100\*(i[1]-d[0]-e[0]-f[0])/(i[1]+g[0])**</formula>

</metric>

* System wide performance metric could be calculated from per-socket events, so the [i] subscript needs to be used in system’s formula tag too. Following is an example from SNB-EP

<metric name=**"metric\_LLC Data Read misses satisfied by remote memory"**>

<event alias=**"a"**>**UNC\_C\_TOR\_INSERTS.MISS\_OPCODE/Match=0x182/NID=0x1**</event>

<event alias=**"b"**>**UNC\_C\_TOR\_INSERTS.MISS\_OPCODE/Match=0x182/NID=0x2**</event>

<formula>**(a[1]+b[0])/(a+b)**</formula>

<formula socket=**"0"**>**b[0]/(a[0]+b[0])**</formula>

<formula socket=**"1"**>**a[1]/(a[1]+b[1])**</formula>

</metric>

**Normalizing Events**

* By default EDP normalizes to events per second which can be misleading when using time as the x-axis and not using a one second sample rate (default sample rate is 0.1 seconds).This option extends EDP to allow control over the interval of time EDP normalizes metrics to.

--normalize TIME

where TIME is the time in seconds to normalize measurements and metrics to.

E.g. --normalize 0.100 would normalize to 100 milliseconds

The pseudo event TSC is also scaled to this time interval and represent the number of TSC clocks in TIME seconds.

**Support for per-transaction metrics**

* If a performance metric can be normalized to transaction in addition to per-instruction, a new tag could be specified to define the per-txn metric name. When the --tps parameter is specified in command line (as part of \*process.cmd), the tool will generate one or more ‘per-txn’ summary view that contains those per-txn metrics. You just need to add the tag “<throughput-metric-name> along with the rest of the details that are used for computing per-instruction metrics

<metric name=**"metric\_DTLB\_MPI"**>

<throughput-metric-name>**metric\_DTLB misses per txn**</throughput-metric-name>

<event alias=**"a"**>**DTLB\_MISSES.WALK\_COMPLETED**</event>

<event alias=**"b"**>**INST\_RETIRED.ANY**</event>

<formula>**a/b**</formula>

</metric>

**Customizing excel charts plotted**

* The tool launches excel automatically to chart various events and metrics. Chart\_format\_<arch>.txt lists a few events and metrics. In case you have a large number of samples collected, excel may take time to draw them. In that case, you can limit the number of entries in this file. There is no requirement that all events and metrics computed need to be graphed. You can also add new entries depending on any new events or metrics that you have defined. A typical file looks like

metric\_DTLB\_MPI

metric\_ITLB\_MPI

If there are no blank lines between two entries, then excel places those charts side-by-side. However, if there are blank lines between entries, the charts are placed one below the other vertically.

* **Multiple metrics in the same chart** : Starting from EDP ver 1.5a, multiple metrics can be plotted in the same chart (previously only one event was plotted per chart). With the new support, you can plot things like CPI and MPI in the same chart
  1. In chart\_\*.txt file, list metrics (that need to appear together in the same chart) in the same line but separated by commas
* **Time stamp on the x-axis for charts**: Prior to ver 1.5a, the x-axis of the charts had only sample numbers. Since each sample is not guaranteed to be for the same duration, it is hard to know the wall clock time associated with any sample. Fortunately, EMON provides –c option (already added to all the collection scripts in this package) that will add time stamp value before every sample. During post-processing, if “—timestamp-in-chart” option (specified in process.cmd) is enabled, then the x-axis of all the charts will contain wall clock time. If EMON data was not collected with –c option, then this option in process.cmd should be commented out. Otherwise, the charts will have blank data.

**Customizing process.cmd**

(Things mentioned here are applicable to jprocess.cmd as well)

* Process.cmd file contains references to metrics definition file and the excel chart definition files. In case a new event file is created (to either support a new architecture or to collect a different set of events than the standard ones provided in this package) or changes made to the events that are to be charted, it is essential to edit process.cmd file and change the following entries

set METRICS=clx-2s.xml

set CHART\_FORMAT=chart\_format\_clx-2s.txt

* Importing dmidecode information: It may be really useful to capture Linux command ‘dmidecode’ output (that describes detailed system information) into the spreadsheet. You can capture dmidecode output in a file and provide that as input for EDP to import into the spreadsheet

set DMIDECODE\_FILE=dmidecode.txt

* Importing general h/w and s/w configuration information: You can create either a text file or another excel (with one sheet) to capture various h/w and s/w information for the workload and import that sheet into EDP spreadsheet so all the information can be in one place.

set CONFIG\_FILE=config.xlsx

* Importing sar and iostat tool outputs: You can import the contents of sar or iostat output or any other tool output by defining the following. The data is not processed but imported as-is

set NETWORKSTAT\_FILE=network.txt

set DISKSTAT\_FILE=disk.txt

* Though you may have collected emon data over a long duration of the workload execution, you may want to analyze only specific portions of the workload execution (either to avoid ramp-up or ramp-down phases or the workload may have several distinct phases of execution). In that case, the tool allows you to post-process a specified range of samples. Currently, you can’t specify this range through wall clock time or clocks elapsed. It has to be based on the sample number. A sample here refers to each EMON sample group. Suppose the collection script has 5 sample groups in each emon data loop. (For example, the first sample in the 2nd emon loop will be referred to as sample #6 for post-processing script. BTW, EMON uses “------“ string to separate individual sample groups). Initially, you can use EDP to post-process the entire data collected. By looking at the time-series plotted cpu clocks and instructions retired excel charts, you can decide on the range of samples that you want to drill down deeper. The following lines in process.cmd need to be modified to process a specific range of samples.

set BEGIN=1

set END=100000

If EMON data is collected with –c option where time stamp for each sample is available, then BEGIN and END samples should provide the time stamp instead of sample numbers as shown below in m/d/yyyy h:mm:ss.000 format.

set BEGIN="08/24/2012 17:53:20.001"

set END="08/24/2012 17:53:35.002"

* By default only the system view will be generated, but user can select one of more of the other views, including socket view, core view and thread view. Note generating core view and thread view makes the processing much slower, depending on the number of cores/threads in your system.

set VIEW="--socket-view --core-view --thread-view"

* As mentioned previously, metrics based on per transactions can also be computed if the throughput per second (TPS) is provided. For example, TPC-E workload would provide the transactions per second. For HPC workloads that have fixed amount of work and measured by elapsed time, you need to come up with a quantity for the work (e.g you may assume 1 million transactions are processed for the entire execution for that workload) and divide that by the elapsed time taken to complete the workload and use that as the TPS value

Set TPS=--tps 1230

**How to Post Process on Linux**

You can install ruby on Linux and directly invoke edp.rb with following command. Excel graphs are produced by edp.rb by calling an external python script, excel\_writer.py included with EDP. Please make sure xlsxwriter is installed: ***pip install XlsxWriter*** before executing the edp script and use the -f option to specify the chart format file if interested in charts.

ruby edp.rb -i <emon-dat-file> -m <XML-metric-file> -o <output-file> --socket-view --core-view –thread-view

Example:

ruby edp.rb -i emon.dat -m clx-2s.xml -o WL\_data\_out --socket-view

Another option is to modify process.sh similar to process.cmd for windows instead of using the edp.rb script directly.

You can get a list of all supported commands with the following command:

ruby edp.rb -h